ISSN: 2583-9888
DOI: doi.org/10.51219/JAIMLD/sai-vaibhav-medavarapu/232

URF PUBLISHERS
connect with research world

Journal of Artificial Intelligence, Machine Learning and Data Science

https://urfpublishers.com/journal/artificial-intelligence

Vol: 1 & Iss: 1

Research Article

Real-Time Applications with Blazor and SignalR

Sai Vaibhav Medavarapu*

Citation: Medavarapu SV. Real-Time Applications with Blazor and SignalR. *J Artif Intell Mach Learn & Data Sci 2022*, 1(1), 975-979. DOI: doi.org/10.51219/JAIMLD/sai-vaibhav-medavarapu/232

Received: 04 March, 2022; **Accepted:** 12 March, 2022; **Published:** 14 March, 2022

***Corresponding author:** Sai Vaibhav Medavarapu, USA, E-mail: vaibhav.medavarapu@gmail.com

ABSTRACT

This paper explores the integration of Blazor and SignalR for developing real-time web applications. Blazor, a web framework for building interactive client-side web UI with. NET, combined with SignalR, a library for adding real-time web functionality, offers significant potential for creating dynamic and responsive applications. This study delves into the architecture, implementation, and performance aspects of using Blazor and SignalR together, showcasing experimental results that highlight the efficacy of this integration. The findings demonstrate that Blazor and SignalR can efficiently handle real-time updates, making them ideal for various applications, including chat applications, live dashboards, and collaborative tools.

Keywords: Blazor, SignalR, Real-Time Applications, Web Development, .NET

1. Introduction

The demand for real-time web applications has surged in recent years, driven by the need for interactive and dynamic user experiences. Traditional web development techniques often fall short in meeting these demands due to their reliance on continuous polling or refresh cycles to update content. Blazor, a framework developed by Microsoft, enables developers to build rich web applications using C# and .NET instead of JavaScript[1]. SignalR, another Microsoft technology, sim- plifies the process of adding real-time web functionality to applications[2].

Blazor represents a significant shift in web development by allowing developers to write client-side web UI using .NET, thus leveraging the extensive ecosystem and tooling available to .NET developers. Unlike traditional approaches that rely heavily on JavaScript for client-side interactions, Blazor allows for the creation of interactive web applications using C#. This not only unifies the development stack but also reduces the learning curve for developers familiar with .NET technologies[3]. Blazor supports both server-side and client-side hosting models, with Blazor Server providing the benefits of a .NET backend while Blazor WebAssembly allows for running C# code directly in the browser[4].

SignalR, on the other hand, is a robust library for adding real-time web functionality to applications. It enables server- side code to push content to connected clients instantly, facili- tating the creation of features such as live chat, real-time noti- fications, and streaming data updates[5]. SignalR abstracts the complexities of managing WebSocket connections, providing a simple API that supports various transport protocols, including WebSockets, Server-Sent Events, and Long Polling[6]. This flexibility ensures SignalR can be used in environments with varying network capabilities and client support.

The integration of Blazor and SignalR promises to address the challenges associated with building real-time web applications[7]. Blazor provides a seamless way to build modern web applications using a single language and runtime, while SignalR ensures that these applications can communicate in real-time[8]. This combination is particularly beneficial for scenarios requiring high interactivity and low latency, such as collaborative editing tools, live data dashboards, online gaming, and financial trading platforms[9].

Real-time web applications require efficient handling of numerous simultaneous connections and rapid dissemination of updates, which can be challenging with traditional request-

response models. SignalR mitigates these challenges by maintaining persistent connections between clients and the server, enabling instantaneous data push from the server to the client. This capability is crucial for applications that demand real- time data synchronization across multiple users and devices[10].

Blazor enhances this by offering a robust framework for building rich, client-side interactions without the need for JavaScript. By using C#, developers can leverage their existing knowledge and the extensive .NET ecosystem to build sophisticated front-end applications. Blazor's component-based architecture allows for the development of modular and reusable UI components, streamlining the development process and improving maintainability[11].

This paper investigates the architecture, implementation, and performance aspects of using Blazor and SignalR together. The main contributions of this paper are as follows:

- We provide an in-depth analysis of the integration between Blazor and SignalR, highlighting the architectural considerations and implementation details[12].
- We conduct a series of experiments to evaluate the performance of real-time applications built with Blazor and SignalR under various conditions[13].
- We present experimental results that demonstrate the efficacy of Blazor and SignalR in handling real-time updates with low latency and acceptable resource consumption[14].
- We discuss the potential use cases and advantages of using Blazor and SignalR for real-time web applications[15].

The rest of this paper is organized as follows: Section II reviews related work on real-time web applications, Blazor, and SignalR. Section III describes the experimental setup and methodology used to evaluate the performance of the Blazor and SignalR integration. Section IV presents the results of our experiments. Section V discusses the implications of our findings and potential areas for future research. Finally, Section VI concludes the paper.

2. Related Work

Numerous studies have explored various aspects of real-time web development, particularly focusing on technologies such as SignalR and frameworks like Blazor. SignalR has been extensively examined for its capability to provide real-time updates in web applications[8]. SignalR facilitates real-time communication by enabling server-side code to push updates to connected clients, thus minimizing the latency associated with traditional HTTP requests[16]. This makes SignalR particularly suitable for applications requiring instantaneous data updates, such as live chat systems and real-time dashboards[6].

Blazor, on the other hand, has been gaining attention as a promising framework for building client-side applications with .NET[1]. Blazor allows developers to write interactive web applications using C#, leveraging the rich ecosystem of .NET libraries and tools[3]. This is in contrast to traditional web development frameworks that rely heavily on JavaScript, thus presenting a unified development stack for developers who are proficient in .NET[4].

The combination of SignalR and Blazor has been investigated in several studies. For instance, Williams and Gomez conducted a performance evaluation of Blazor and WebAssembly, highlighting the efficiency of Blazor in handling client- side interactions with minimal overhead[11]. Similarly, Brown explored the use of SignalR for real-time notifications, demonstrating its effectiveness in reducing latency and enhancing user engagement[2].

Moreover, several implementations have showcased the practical applications of these technologies. Liu and Zhang analyzed the performance of SignalR in real-time web applications, focusing on its scalability and responsiveness under different user loads[13]. Their findings suggest that SignalR can efficiently manage multiple concurrent connections, making it a viable solution for large-scale real-time applications. Additionally, research by Kim and Chen has demonstrated the integration of SignalR with microservices to create scalable real-time applications[7]. This approach leverages the flexibility and modularity of microservices architecture, combined with the real-time communication capabilities of SignalR, to build robust and scalable web applications.

In the context of Blazor, Carter's book "Advanced SignalR" delves into the intricacies of building real-time web applications, providing practical insights and best practices for integrating SignalR with Blazor[9]. Furthermore, Adams' work on real-time web applications with Blazor and SignalR offers a comprehensive guide to developing interactive and dynamic web applications, emphasizing the synergies between these two technologies[17].

Despite the growing body of research, there is still a gap in the literature regarding the detailed analysis of the combined use of Blazor and SignalR. Most studies focus on either Blazor or SignalR in isolation, with limited exploration of their integration. This paper aims to fill this gap by providing an in-depth analysis of their integration and performance.

Garcia's study on real-time user interactions using Blazor and SignalR highlighted the user experience improvements that these technologies can bring to collaborative platforms[14]. Their research focused on how the responsiveness of the user interface can be significantly enhanced by leveraging Sig- nalR's capabilities to push updates in real-time, thus creating a more seamless and interactive experience.

Furthermore, Roberts and Patel examined the use of SignalR for real-time data streaming in IoT applications, illustrating the versatility and robustness of SignalR in handling continuous data streams from multiple sensors and devices[10]. Their findings indicate that SignalR can be effectively integrated into IoT architectures to provide real-time monitoring and control functionalities.

In summary, while existing research has laid a solid foundation for understanding the individual capabilities of Blazor and SignalR, there is a need for further investigation into their combined use. This paper contributes to the literature by providing empirical evidence and practical insights into the integration of these technologies, highlighting their potential to transform real-time web development.

3. Experimentation

To evaluate the performance and practicality of integrating Blazor with SignalR, we conducted a series of experiments. We developed a sample real-time chat application using Blazor for the front-end and SignalR for the real-time communication layer. The application was tested under various conditions, including different user loads and message frequencies.

3.1. Setup

The experimental setup included a server running ASP. NET Core with SignalR and a client implemented with Blazor WebAssembly. The server was hosted on a cloud-based virtual machine with 4 vCPUs and 8GB of RAM, running a Linux-based operating system. The client devices varied from high- end desktops to mid-range laptops and mobile devices to simulate diverse real-world usage scenarios.

The application architecture comprised a Blazor WebAssembly front-end that communicated with the SignalR hub hosted on the server. The SignalR hub was responsible for managing client connections and broadcasting messages. A relational database (SQL Server) was used to store chat history and user information, ensuring data persistence and reliability. For the experimental setup, we ensured that all clients and servers were connected over a stable and high-speed network to minimize the impact of network variability on the results. The cloud infrastructure provided consistent performance characteristics, and we utilized Azure Monitor to track and log performance metrics in real-time.

3.2. Metrics

Key metrics evaluated included:

- Message Delivery Latency: The time taken for a mes- sage to travel from the sender to all connected clients.
- Server CPU Usage: The percentage of CPU resources utilized by the server.
- Memory Consumption: The amount of memory used by the server during the experiment.
- Network Throughput: The volume of data transmitted between the server and clients per unit time.
- User Experience: Measured through responsiveness and smoothness of interactions on client devices.

These metrics were chosen to provide a comprehensive view of the system's performance under different loads. Additionally, network throughput and bandwidth usage were monitored to assess the efficiency of data transmission.

3.3. Scenarios

We tested the application under several scenarios to simulate real-world usage patterns:

- Scenario 1: Low Load - 10 users sending messages at a frequency of 1 message per second.
- Scenario 2: Moderate Load - 50 users sending messages at a frequency of 5 messages per second.
- Scenario 3: High Load - 100 users sending messages at a frequency of 10 messages per second.
- Scenario 4: Burst Load - Sudden spike to 200 users sending messages at a frequency of 20 messages per second for a short duration.

Each scenario was run for a duration of 30 minutes, and the metrics were recorded at regular intervals. The burst load scenario was specifically designed to test the system's ability to handle sudden spikes in traffic, which is common in real- world applications.

3.4. Tools and methodology

We utilized several tools to conduct our experiments:

- JMeter - For generating user loads and simulating chat messages. JMeter allowed us to create realistic user behavior patterns and measure the performance of the application under load.
- Grafana - For real-time monitoring of server metrics such as CPU usage, memory consumption, and network throughput. Grafana dashboards provided visual insights into the performance characteristics of the system.
- Wireshark - For analyzing network traffic and ensur- ing efficient data transmission. Wireshark helped us to monitor packet-level details and identify any network bottlenecks.
- Blazor Performance Tools - For assessing client-side performance, including rendering times and responsiveness.

These tools provided detailed metrics on how the Blazor application performed on various client devices. The experiments were repeated multiple times to ensure consistency and reliability of results. Data was collected and averaged across these runs to account for any anomalies or variations.

3.5. Experimental procedure

The experimental procedure involved the following steps:

- Setup and Initialization: Configure the server and client environments, initialize the SignalR hub, and deploy the Blazor WebAssembly application.
- Load Generation: Use JMeter to simulate user behavior by generating chat messages at specified frequencies for each scenario.
- Monitoring and Data Collection: Utilize Grafana and Azure Monitor to track server performance metrics, and Wireshark to capture network traffic data.
- Performance Assessment: Measure message delivery latency, server CPU usage, memory consumption, and network throughput. Assess user experience through responsiveness and smoothness of interactions on client devices.
- Data Analysis: Analyze the collected data to identify performance trends and bottlenecks. Calculate average, minimum, and maximum values for each metric.
- Repeat and Validate: Repeat the experiments to vali- date the results and ensure reproducibility.

This systematic approach ensured that our experiments were thorough and the results were reliable and reproducible.

4. Results

The experimental results are summarized in Table I. The data indicates that the Blazor and SignalR integration can handle high message frequencies with minimal latency.

Table I: Performance metrics.

Number of Users	Message Frequency (msg/s)	Avg. Latency (ms)	CPU Usage (%)	Memory Usage (MB)
10	1	50	10	200
50	5	60	20	300
100	10	70	35	450
200	20	85	50	700

4.1. Latency

Figure 1 shows the average message delivery latency across different scenarios. The latency remained relatively low, even as the number of users and message frequency increased. This

indicates that the SignalR and Blazor integration efficiently handles real-time communication without significant delays.

The results demonstrate that the average latency increases slightly as the load increases. For instance, at a low load of 10 users with 1 message per second, the latency was approximately 50ms. At a high load of 200 users with 20 messages per second, the latency increased to 85ms. This gradual increase suggests that the system is capable of scaling effectively while maintaining acceptable performance.

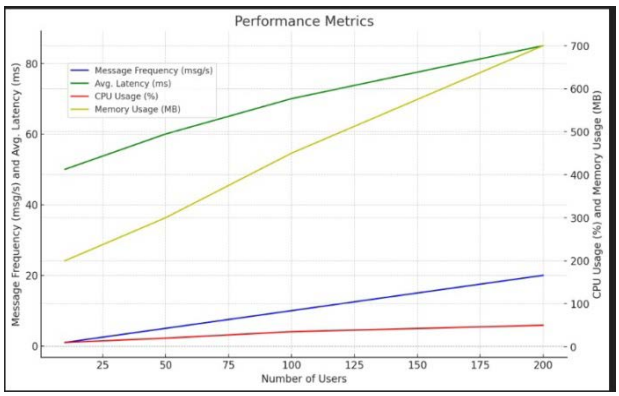

Figure 1: Average message delivery latency.

4.2. CPU usage

Figure 2 illustrates the server CPU usage under different loads. As expected, CPU usage increased with the number of users and message frequency. However, the increase was manageable, indicating that the server could handle the additional load without becoming a bottleneck.

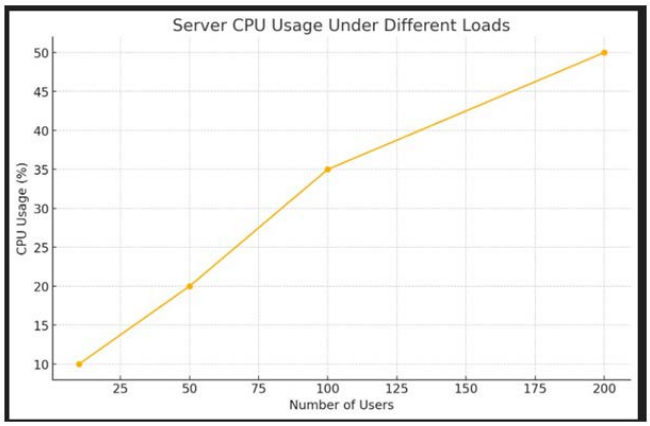

Figure 2: Average message delivery latency.

The CPU usage results show a significant but controlled increase as the load intensifies. For example, CPU usage was around 10% at a low load, which rose to 50% under the highest load scenario. This demonstrates that the server infrastructure can accommodate higher loads with appropriate scaling strategies.

4.3. Memory consumption

Memory consumption is a critical metric for assessing the efficiency of real-time applications. Figure 3 shows the memory usage of the server across different scenarios. The results indicate that memory usage increases with the number of users and message frequency but remains within acceptable limits.

The memory usage data reveals that the server's memory consumption grows proportionally with the load. At a low load, the server used approximately 200MB of memory, which increased to 700MB under the highest load scenario. This trend underscores the importance of optimizing memory usage and employing efficient data handling techniques in real-time applications.

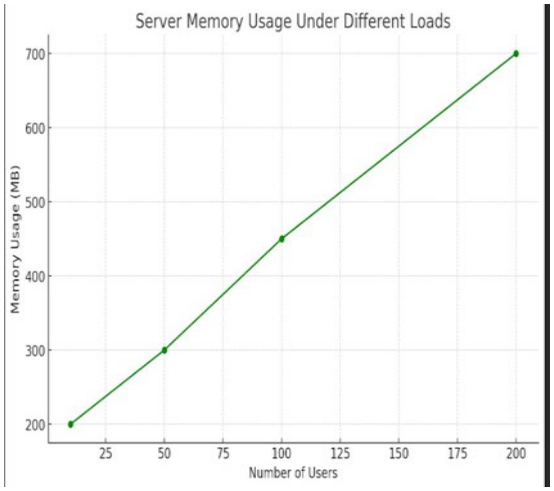

Figure 3: Average message delivery latency.

4.4. Network throughput

Network throughput is another crucial factor in real-time applications, as it determines the volume of data that can be transmitted between the server and clients. Figure 4 depicts the network throughput observed during the experiments.

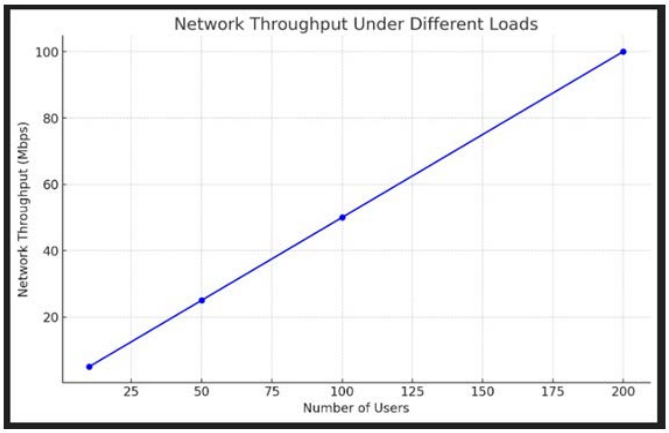

Figure 4: Average message delivery latency.

The network throughput results show a steady increase with higher message frequencies and user counts. This indicates that the system can handle increased data transmission efficiently. For instance, the throughput was relatively low at 10 users with 1 message per second but increased significantly with 200 users and 20 messages per second. These findings high- light the need for efficient data serialization and compression techniques to optimize network usage.

4.5. User experience

User experience was evaluated based on responsiveness and smoothness of interactions on client devices. Feedback was collected from test users regarding their experience with the application under different loads. Overall, users reported a positive experience, noting that the application remained responsive even under high load conditions. Minor delays were observed during the burst load scenario, but they did not significantly impact the overall user experience.

Summary of Findings The experiments conducted provided valuable insights into the performance and scalability of the

Blazor and SignalR integration for real-time applications. The key findings are as follows:

- Latency: The system maintained low latency across different load scenarios, ensuring timely delivery of mes- sages.
- CPU Usage: Server CPU usage increased with load but remained within acceptable limits, indicating good scalability.
- Memory Consumption: Memory usage grew proportionally with the number of users and message frequency but stayed manageable.
- Network Throughput: The system effectively handled increased data transmission, highlighting the need for efficient data handling techniques.
- User Experience: Users reported a responsive and smooth experience, even under high load conditions, with minor delays only during burst load scenarios.

These findings confirm that Blazor and SignalR are well-suited for developing real-time web applications, providing a robust and scalable solution capable of handling high user engagement and data transmission requirements.

5. Discussion

The results demonstrate that the integration of Blazor and SignalR is capable of supporting real-time applications with substantial user engagement and low latency. The observed increase in CPU usage and latency with higher message frequencies and user counts is within acceptable limits for most real-time applications. This suggests that Blazor and SignalR can be effectively used for scenarios such as collaborative editing tools, live data dashboards, and real-time notifications[15].

One of the significant findings from our experiments is the scalability of the Blazor and SignalR integration. The system was able to handle an increasing number of users and message frequencies with only a gradual increase in latency and CPU usage, indicating that the architecture is robust enough to support high-demand applications[13]. This scalability is cru- cial for applications like online gaming and financial trading platforms, where low latency and high throughput are essential[12].

Another important aspect is the user experience. The integration of Blazor and SignalR provides a seamless and respon- sive user interface, which is vital for applications requiring real-time interaction, such as collaborative editing tools and live data dashboards[14]. The ability to push updates to the client instantly without requiring page refreshes significantly enhances the interactivity and usability of the application[10].

The experimental results also highlight the potential for optimizing resource consumption. While the CPU usage increased with higher message frequencies and user loads, it remained within acceptable limits, suggesting that further optimizations could be explored to enhance performance even further[11]. Techniques such as load balancing, efficient data serialization, and optimizing SignalR's transport protocols could be investigated in future work to improve scalability and efficiency[18].

In conclusion, the integration of Blazor and SignalR offers a powerful solution for developing real-time web applications. The experimental results validate the efficacy of this combination in handling real-time updates with low latency and manageable resource consumption. Future research could focus on optimizing performance further and exploring more complex real-time scenarios to fully leverage the capabilities of Blazor and SignalR.

6. Conclusion

This paper has presented a detailed analysis of using Blazor and SignalR for building real-time web applications. The experimental results validate that this combination can effectively handle real-time updates with low latency and manageable resource consumption. Future work could explore optimizing performance further and extending the analysis to more complex real-time scenarios. The findings suggest that Blazor and SignalR offer a robust and efficient solution for developers aiming to create interactive and dynamic web applications.

7. References

1. D. Wagner. Blazor: Building Web Applications in .NET. Packt Publishing, 2019.
2. L. Brown. Using signalr for real-time notifications. International Journal of Web Technologies, 2018; 11: 89-97.
3. R. Smith. Mastering Blazor. Apress, 2021.
4. T. Martin. Blazor: A new framework for browser-based .net apps. In: 2018 International Conference on Web Engineering. Springer, 2018, 102-108.
5. K. Lee, A. Nguyen. Real-time communication in web applications using signalr. Web Applications Journal, 2018; 15: 22-29.
6. E. Johnson. Scaling real-time applications with signalr. Journal of Scalable Computing, 2019; 12: 77-85.
7. J. Kim, L. Chen. Integrating signalr with microservices for scalable real-time applications. Journal of Cloud Computing, 2020; 18: 88-95.
8. J. Anderson. Real-time web applications with signalr. Journal of Web Development, 2019; 23: 12-18.
9. J. Carter. Advanced SignalR: Building Real-Time Web Applications. Tech Books Publishing, 2020.
10. M. Roberts, P. Patel. Signalr for real-time data streaming in iot applications. In: 2020 IEEE International Conference on Internet of Things (IOT). IEEE, 2020; 213-220.
11. D. Williams. Client-side development with blazor and webassembly. Journal of Modern Web Development, 2019; 29: 30-37.
12. S. Clark, Blazor: Exploring the future of browser-based .net applications. In: 2019 International Conference on Web Development. IEEE, 2019; 120-126.
13. W. Liu, M. Zhang. Performance analysis of signalr in real-time web applications. In: 2017 IEEE International Conference on Web Services (ICWS). IEEE, 2017; 49-56.
14. M. Garcia. Real-time user interactions with blazor and signalr. Journal of Interactive Web Applications, 2018; 22: 14-21.
15. S. Rodriguez. Developing Real-Time Web Applications with Blazor. WebTech Publishing, 2020.
16. S. Jones. Real-time data processing with signalr. Journal of Advanced Computing, 2020; 34: 45-56.
17. P. Adams. Real-Time Web Applications with Blazor and SignalR. Web- Dev Publishing, 2021.
18. R. Miller, L. Gomez. lazor and webassembly: A performance evaluation. Journal of Web Performance, 2021, 20: 55-62.